Smart city traffic light

Ismail Elsayed
Mirtualation Number: 7219301
Dortmund, Germany
ismail.elsayed001@stud.fh-dortmund.de

Raed Kayali
Mirtualation Number: 7219191
Dortmund, Germany
raed.kayali001@stud.fh-dortmund.de

Hadis Mohammadi
Mirtualation Number: 7219036
Dortmund, Germany
Hadis.mohammadi005@stud.fh-dortmund.de

Amr Abdelaziz
Mirtualation Number: 7216843
Dortmund, Germany
amr.abdelaziz003@stud.fh-dortmund.de

Anguiga Hermann
Mirtualation Number:
Dortmund, Germany
hermann.anguiga@stud.fh-dortmund.de

***Abstract*— The Smart City system for networked traffic control is an intelligent and interconnected infrastructure that leverages advanced technologies to optimize traffic flow and enhance transportation efficiency in urban areas. This system integrates various components, including traffic sensors, communication networks, data analytics, and control algorithms, to create a dynamic and responsive traffic management solution. By collecting real-time data on traffic conditions, the system can make informed decisions and implement adaptive control strategies such as signal timing adjustments, traffic rerouting, and prioritization of emergency vehicles. Through continuous monitoring, analysis, and optimization, the Smart City system aims to alleviate congestion, reduce travel times, improve safety, and enhance the overall mobility experience for residents and visitors. This paper proposes the modeling of Smart city, detailed Modeling and design of smart traffic light using UML different diagrams. Then there is detailed implementation, testing and scheduling for the system using Tinker-cad, google testing framework for cpp and python script code.**

Keywords—Smart City, Smart Traffic Light, Smart Traffic Management.

I. Introduction

In the domain of urban planning, the idea of smart cities is gaining traction as it seeks to improve city life through technological advancements. An integral part of this pursuit involves enhancing transportation systems, particularly through the implementation of intelligent traffic light systems. This paper specifically explores the design, development, and implementation of a Smart Traffic Light System (STLS) within the broader framework of smart city initiatives.

SysML, known as Systems Modeling Language, provides a structured approach for conceptualizing and building complex systems like smart traffic light networks. Through SysML, stakeholders can effectively model various elements of traffic management systems, ensuring they meet stakeholder needs and adhere to established design standards. The STLS architecture incorporates crucial hardware and software components, including advanced sensors, adaptive control algorithms, and real-time communication protocols.

The integration of the Smart Traffic Light System (STLS) with other smart mobility projects plays a crucial role in creating a cohesive urban transportation network focused on efficiency, sustainability, and user satisfaction. By incorporating behavior modeling, the STLS can simulate various traffic scenarios, evaluate its performance, and adjust traffic flow to enhance overall efficiency and safety. This holistic approach aims to build a smooth urban transportation infrastructure that not only tackles existing challenges but also anticipates and adapts to future needs, ultimately leading to the development of smarter, more livable cities.

II. Design Approach

A. SysML Overview

According to [1], SysML serves as a visual language that expands upon the Unified Modeling Language (UML). It comprises nine types of diagrams grouped into three categories: Behavior, Requirement, and Structure. Each diagram offers a specific view of a system's design when applied in practice.

In system design, SysML diagrams act as specialized tools, providing different perspectives on a system's architecture and behavior. Behavior Diagrams illustrate how a system operates dynamically over time, while Requirement Diagrams manage and document the system's requirements to ensure they meet stakeholder needs. Structure Diagrams, in contrast, offer a static representation of the system's components and their relationships.

B. SysML in Smart City Design

In smart city design, SysML (Systems Modeling Language) plays a crucial role in conceptualizing and developing the complex systems that form the backbone of urban infrastructure and services. By using SysML's graphical language and various diagram types, designers can effectively model and analyze different aspects of smart city systems, including transportation, energy management, waste disposal, and public safety.

SysML allows stakeholders to understand the complex connections and interactions among various components within a smart city ecosystem. Behavior Diagrams, like Activity Diagrams and State Machine Diagrams, help visualize dynamic system behaviors, such as traffic optimization algorithms or energy consumption patterns. Requirement Diagrams provide a structured way to define and manage the diverse requirements from stakeholders and regulations, ensuring smart city solutions meet community needs.

Additionally, Structure Diagrams, such as Block Definition Diagrams and Internal Block Diagrams, offer insights into the architectural composition of smart city systems, illustrating

how subsystems, components, and interfaces relate to each other. This aids in identifying potential issues early on, like scalability problems or potential points of failure, to create resilient urban infrastructure.

C. Design Process

1) Requirements Analysis

Requirements are generally the starting point for many engineering designs. Here we will also follow that principle. A smart city design is a very complex entity simply because it has many requirements. Fortunately, for this design the focus would be only on a few.

In Fig. 1, we can see a snippet of some requirements. We could notice that each requirement has an id and falls under the ten main requirements which are the foundation (basis) of this work. in this section, we will only cover three of them.

The design of this Smart City shall include strong Communication Protocols. This refers to the essential standards and frameworks that regulate the exchange of data and information among diverse devices, sensors, and systems embedded in the city's infrastructure. These established protocols play a crucial role in ensuring the smooth and effective communication between the various components of the smart city ecosystem, thereby promoting interoperability and integration. Examples of such protocols include MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), and HTTP (Hypertext Transfer Protocol). The Smart city system must also optimize traffic algorithms. Traffic optimization algorithms are computational methods designed to improve the flow of vehicles and pedestrians within urban environments. These algorithms utilize various data sources, such as traffic sensors, GPS data, and historical traffic patterns, to analyze and predict traffic conditions. They then generate optimized routes, signal timings, and other interventions to alleviate congestion, reduce travel times, and enhance overall transportation efficiency. Examples of traffic optimization algorithms include traffic signal optimization. The Smart City shall have User Interface. User Interface (UI) refers to the graphical or visual elements that enable interaction between residents, city administrators, and the various technologies deployed within the urban environment. These interfaces can take many forms, including mobile applications, web portals, interactive kiosks, and digital signage. The goal of a smart city UI is to provide intuitive and user-friendly access to information and services related to transportation, public safety, utilities, governance, and more.

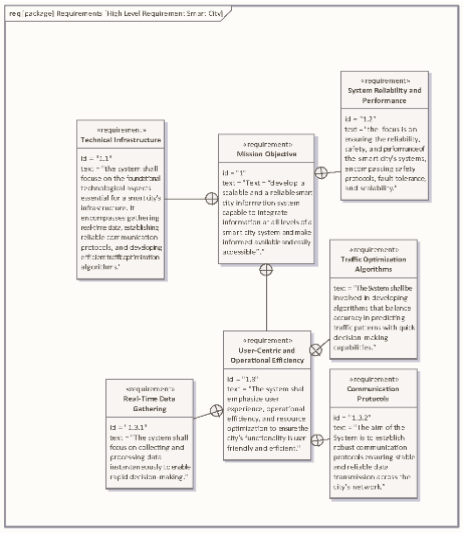


Fig. 1. High Level Requirement Smart City

2) System Architecture Design

In the modern age of technology, cities are transforming into lively environments driven by interconnected digital advancements, leading to the emergence of Smart City Systems. These systems combine different aspects of city living, such as the economy, healthcare, infrastructure, and transportation, to promote sustainable development, effectiveness, and the well-being of citizens. Now, let's explore the structural blueprint of a Smart City System, emphasizing the essential components that facilitate its operation and creativity.

Fig. 2 depicts the functional architecture of the Smart City System defined in a SysML block definition diagram (BDD). It demonstrates the expected level of detail within the system, including its content and outcomes, facilitating communication among all Smart City subsystems [2]. The Smart Economy block lies at its center. It coordinates a dynamic array of digital platforms, financial systems, and inventive business models. By utilizing technologies such as blockchain, AI-powered analytics, and IoT-linked supply chains, this block promotes entrepreneurship, facilitates job generation, and encourages sustainable development. Revolutionizing medical services, the Smart Healthcare block focuses on preventive care, personalized treatments, and smooth patient experiences. With telemedicine platforms, wearable tech, and predictive analytics, it enables proactive health monitoring and early intervention. Integrated electronic health records streamline data exchange among providers, optimizing care coordination and improving patient outcomes. Serving as the backbone of the Smart City System, the Smart Infrastructure block encompasses resilient networks, sustainable buildings, and efficient transportation systems. Utilizing IoT sensors, predictive maintenance algorithms, and real-time monitoring, this block enables proactive infrastructure management, minimizing downtime and enhancing safety. Smart grids and energy-efficient buildings reduce carbon footprints, while intelligent transportation systems optimize traffic flow and reduce congestion. Transforming urban transport, the Smart Mobility block ensures accessibility, sustainability, and safety. Through interconnected autonomous vehicles, electric fleets, and shared services, it offers seamless transportation. Advanced

traffic management and predictive analytics reduce commute times and carbon emissions. Utilizing data-driven insights and technology solutions, the Smart Environment block monitors and manages natural resources, mitigates pollution, and promotes environmental sustainability through initiatives like smart waste management and air quality monitoring networks. The Smart Governance block digitizes government services and implements open data initiatives. Similarly, the Smart Education block empowers lifelong learning through digital platforms and immersive educational experiences. The Smart People block emphasizes human-centric design principles, community empowerment, and social inclusion, fostering collaboration and creativity among residents. Incorporating digital infrastructure and innovation ecosystems, the Smart Technology block guarantees the scalability, interoperability, and resilience of smart city solutions through collaboration among academia, industry, and government.

A vital component of this model is the integration of an information platform, serving as a gateway for effortless information access within a smart city and enabling interaction among all smart subsystems of an SC system. In this project's context, data is generated and processed into meaningful information at the level of each smart subsystem, as depicted in Fig. 3.

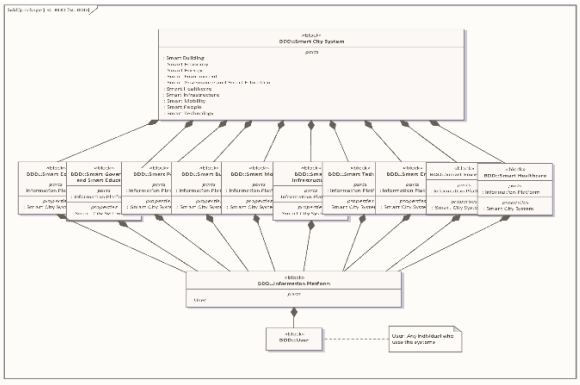


Fig. 2. Smart City Functional Architecture

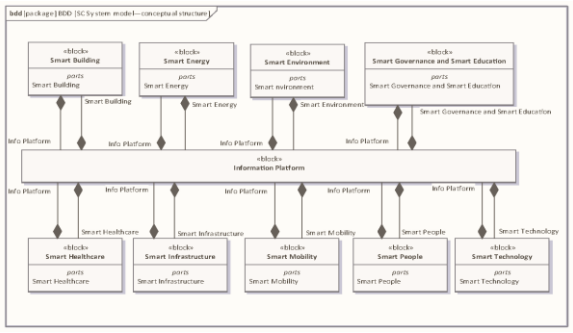


Fig. 3. Smart City System model—conceptual structure

For this project, we will focus on Smart Mobility. The Smart Mobility Module comprises three essential sub-systems: Smart Traffic Lights, Public Transportation Optimization, and Intelligent Transport Systems (Fig 4). Smart Traffic Lights utilize real-time data analytics and machine learning to dynamically adjust signal timings, reducing congestion and prioritizing pedestrian safety. Public Transportation Optimization employs predictive analytics to adjust routes and schedules, enhancing efficiency and encouraging modal shift towards sustainable transit options. Intelligent Transport Systems integrate GPS tracking and communication networks to monitor and manage transportation assets, facilitating seamless integration and improving overall mobility efficiency. Together, these sub-systems work in harmony to transform urban transportation, making it safer, more efficient, and environmentally sustainable. The Smart Mobility Module faces constraints in real-time signal adjustments, data privacy, and interoperability. Infrastructure limitations, privacy concerns, and integration challenges hinder its ability to effectively manage traffic flow and optimize public transit. Overcoming these constraints is crucial for realizing the module's potential in enhancing urban transportation efficiency and sustainability.

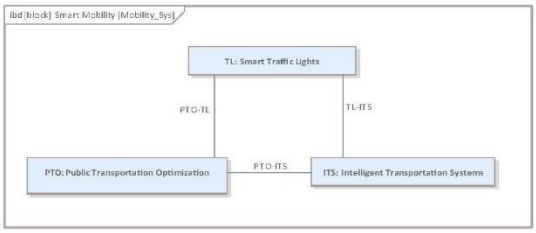


Fig. 4. Smart Mobility System

3) Behavior Modeling

a) Use case diagram.

- Smart vehicle connection

In the Smart Vehicle Use Case Diagram as depicted in Fig. 5, the driver plays a central role, interacting with various integrated functionalities aimed at improving safety, efficiency, and convenience during travel. These functionalities rely heavily on robust data connectivity services, enabling seamless communication and interaction between the vehicle and its surroundings.

Safety and emergency services are paramount in this ecosystem. Equipped with advanced sensors and communication technologies, the smart vehicle can identify potential hazards and take proactive measures to alert the driver or mitigate risks. In emergency situations like accidents or medical emergencies, the vehicle automatically contacts emergency services, providing precise location data and pertinent information for prompt assistance.

Moreover, the smart vehicle utilizes data connectivity to provide comprehensive navigation and traffic assistance, parking guidance, and maintenance diagnostics. Drivers benefit from real-time navigation updates, optimized route suggestions, and assistance in finding parking spots. Maintenance diagnostics ensure vehicle health through continuous monitoring and timely alerts for servicing needs. Additionally, robust privacy and security measures are implemented to safeguard sensitive data and fend off cyber threats, ensuring a secure driving experience for both the driver and passengers

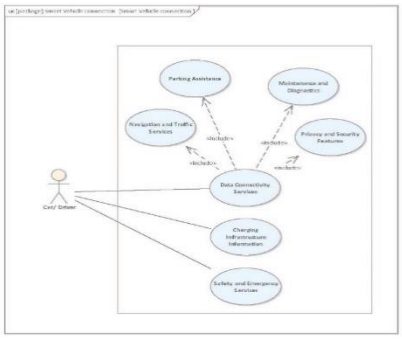


Fig. 5. Smart Vehicle Connection

- Smart parking

The Smart Parking Management Use Case Diagram depicts the interactions between various actors and functionalities within the system, including residents, the Smart City System, government agencies, and IoT devices/sensors (Fig. 6).

Residents, as primary users, engage with the system to find available parking spaces, reserve spots via a mobile app, and access relevant data. The Smart City System acts as the central hub, coordinating parking information, providing services, and facilitating communication between residents, government agencies, and IoT devices/sensors.

Government agencies play a crucial role in managing parking regulations, enforcing policies, and overseeing the overall operation of the parking management system. IoT devices and sensors are deployed throughout the city to monitor parking spaces' availability and relay real-time data to the Smart City System.

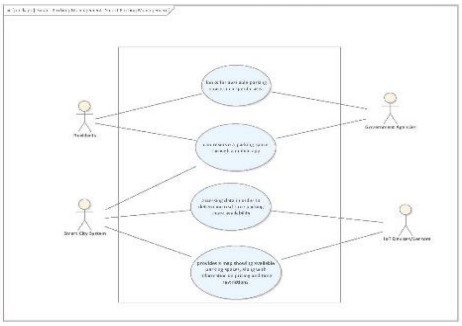


Fig. 6. Smart Parking Management

b) Context diagram.

The Smart City system is not a borderless entity. Its boundaries and scope within the broader urban environment are illustrated in the Context Diagram in Fig. 7. This clarification is crucial for establishing a clear focus for planning, implementing, and managing smart city projects, ensuring that resources are allocated effectively, and objectives are aligned with the system's capabilities and limitations. Moreover, this context diagram illustrates the interactions and dependencies between the smart city system and its external environment, including residents, businesses, and government agencies, highlighting the boundaries where data, services, and responsibilities are shared.

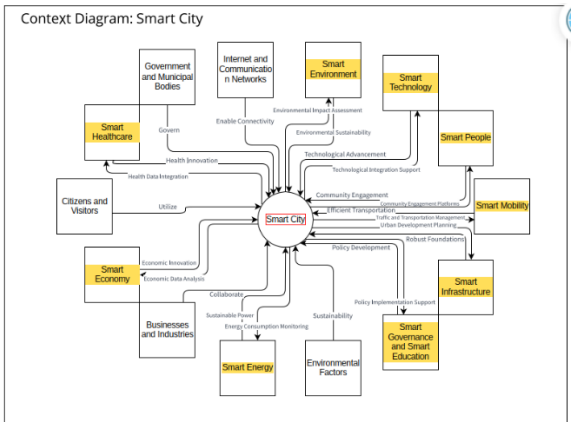


Fig. 7. Smart City Context Diagram

III. Design of Smart Traffic Light System

A. *System Architecture*

The Traffic Light Subsystem Block Definition Diagram (BDD) in Fig. 8 outlines the essential parts needed to manage urban traffic effectively. These components include Sensor Integration, which gathers real-time data from various sources like vehicle detectors and pedestrian sensors, allowing adjustments to traffic light timings based on current conditions. The Connectivity Interface ensures smooth communication with external networks, aiding interoperability with smart city infrastructure and emergency services for efficient coordination and data exchange. Power Management oversees energy distribution and optimization strategies, ensuring reliable and sustainable traffic light operations, even during power outages or adverse conditions. Additionally, the Emergency Response Integration block integrates emergency capabilities directly into the traffic light infrastructure, enabling quick responses to accidents or medical incidents. Through direct access to emergency services and prioritization features for emergency vehicles, this block enhances safety and minimizes response times, essential for mitigating the impact of critical events on urban mobility. Together, these components create a cohesive system that improves traffic flow, safety, and overall efficiency within the smart city infrastructure.

Fig. 9 is a sysML internal block definition diagram of the traffic light subsystem. It illustrates a complex network of components designed for efficient traffic management and emergency response. The Sensor Integration Module combines vehicle and pedestrian detection sensors with a Data Processing Unit to gather and process real-time traffic data, informing decisions on light timings. Within the Traffic Light block, components like the Timing and Sequence Algorithm, Emergency Override system, and Traffic Controller ensure effective light control. The Emergency Response Integration module interfaces with emergency services, enabling swift responses to incidents through priority signal overrides and automated alerts. Connectivity facilitates real-time data transmission with external networks, while the Power Management System ensures reliable

operation with power distribution and backup supply. Together, these components form a cohesive system, optimizing traffic flow and enhancing safety in the smart city environment.

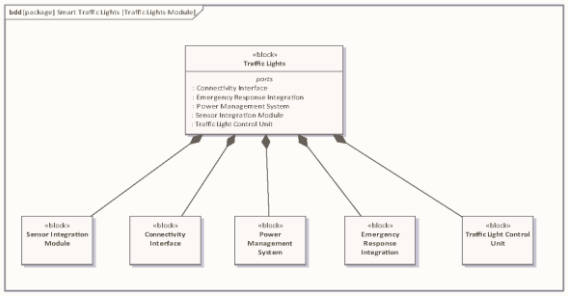


Fig. 8. Traffic Lights Module

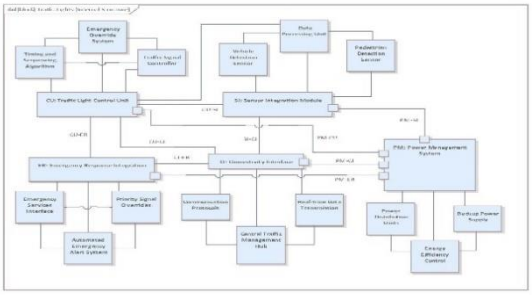


Fig. 9. Traffic Light Internal Structure

B. Key Components

a) Traffic Light Controllers (Arduino): These devices serve as the brains of the traffic light system, controlling the timing and sequencing of the lights based on inputs from sensors and external factors. We used Arduino Uno boards for their flexibility and programmability.

b) Sensors : These sensors include traffic sensors embedded in roadways to monitor vehicle flow, pedestrian sensors, and buttons to detect pedestrians waiting to cross.

c) Communication Infrastructure: This component facilitates communication between different elements of the traffic light system, such as the controllers, sensors, and central management system. It enables real-time data exchange and coordination to ensure efficient traffic flow and responsiveness to changing conditions.

d) Power Supply: Reliable power sources are essential to ensure uninterrupted operation of the traffic light system. The Arduino Uno is powered by a 9v battery and then the system Is powered by the Uno.

e) Lights: The lights themselves are a critical component of the traffic light system, providing visual signals to drivers and pedestrians. We used LED commonly found in electronic projects.

C. System Behavior

1) State Machine Diagrams

The traffic controller state machine operates with four states (Fig. 10): two Yellow/Orange states, one Red state, and a Green state, which acts as a super state. Within the Green state, there are two sub-states: Normal Operation State and Emergency Override State. Transitions between states occur upon receiving a signal, indicated by sig = 1 or true. If there's no signal (sig = 0), the state remains unchanged.

Initially, the traffic controller starts in the Green state's Normal Operation State, allowing traffic flow. When a signal is received (sig = 1), the machine moves to the Yellow/Orange state, signaling caution. After another signal, it shifts to the Red state, indicating a stop. Then, with another signal, it returns to the Green state, resuming normal operation.

In emergencies, if an emergency signal is received (sig = 1), the machine switches from the Green state's Normal Operation State to the Emergency Override State. Here, the traffic controller prioritizes emergency vehicles or manages urgent situations. Once the emergency ends or the signal stops (sig = 0), the machine goes back to the Green state's Normal Operation State, continuing regular traffic control.

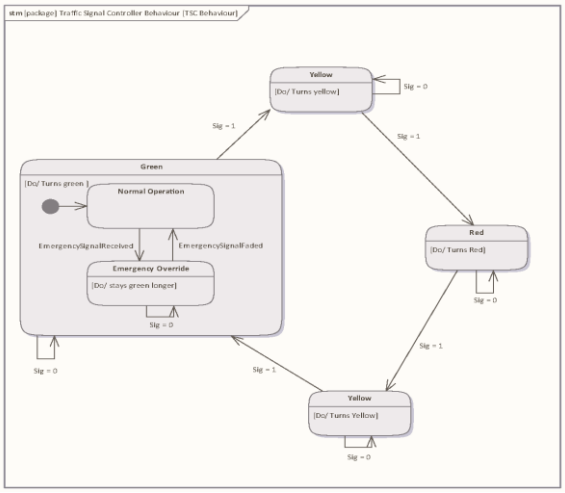


Fig. 10. Smart Traffic Light Behaviour

2) *Activity Diagrams:*

The activity diagrams presented depict the operation of a smart traffic light system in a smart city using SYSML. The diagrams cover three different scenarios: a car, a pedestrian, and an emergency vehicle attempting to cross the road. These diagrams are described as follows:

1- As shown in figure 11 the activity diagram for a car waiting requesting to cross, it is described as follows: The process begins with a car requesting to cross the road, which triggers the green light. This request is received by the traffic control unit, which sends a signal to the server. At the same time, the system checks for an emergency state and the timing of other routes. If the time is not equal to the waiting time and there is no emergency state, a signal is sent to ensure that the traffic lights for other crossing routes turn red. This allows the requested path to be given a green light, enabling the car to cross the road safely. This process ends when the car has safely crossed the road. The activity diagram ensures efficient and safe traffic management in a smart city.

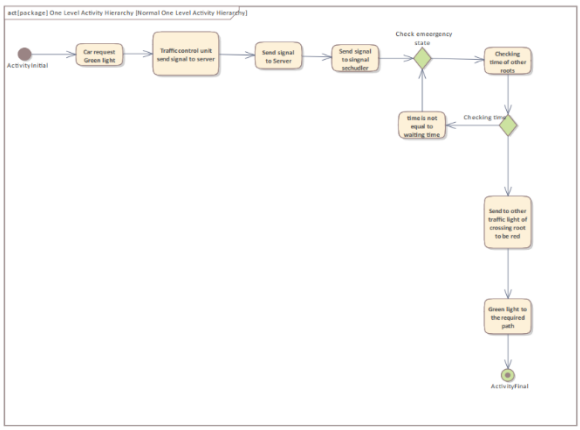
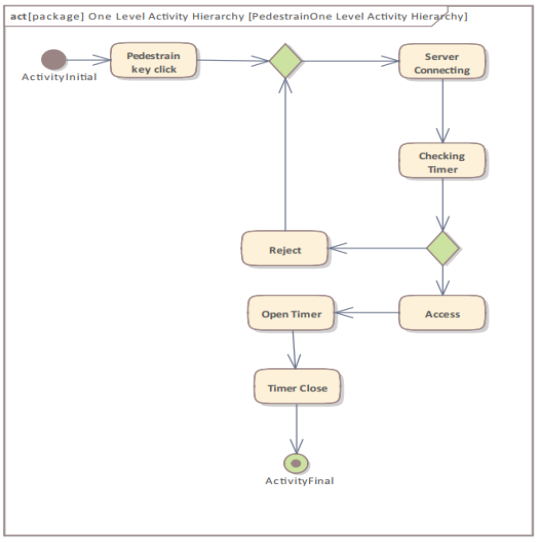


Fig. 11: Showing Normal car send request to traffic light to cross the road

2- In the case of a pedestrian trying to cross the road the activity diagram as shown in Fig. 12 can be described as follows: The process begins with the pedestrian clicking a key. This action initiates a server connection that triggers a checking timer. If the timer is not approved, it leads to rejection, and the pedestrian must wait. If approved, an open timer is activated granting access for the pedestrian to cross safely. The process ends when the timer closes after the pedestrian has safely crossed the road.



Figu. 12: Showing pedestrian request to traffic light to cross the road

3- The activity diagram for a smart traffic light system as shown in Fig. 13, in the case of an emergency car trying to cross the road, can be described as follows: The process begins with the emergency car requesting passage. This request is received by the traffic light, which prioritizes it due to the emergency car having the highest priority in the system. If there is a pedestrian close by, their crossing is halted to allow safe passage for the emergency vehicle. The server then sends a root command to grant access to the emergency car by turning the traffic light green. This ensures that the emergency car can cross swiftly and safely. The process ends when the emergency car has safely crossed the road.

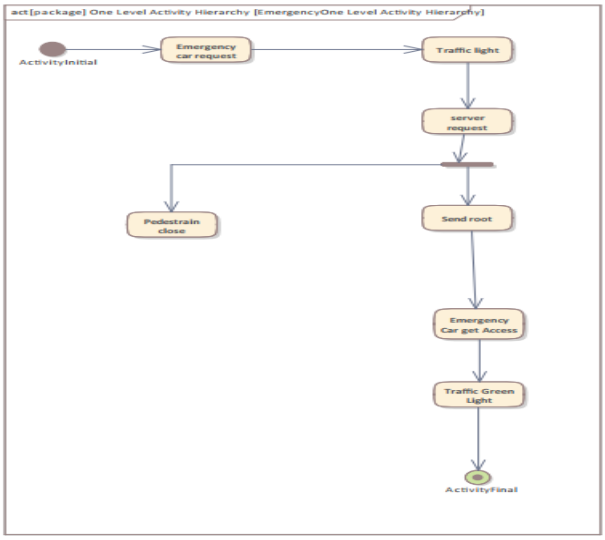


Fig. 13: Showing emergency car request to traffic light to cross the road

3) Sequence Diagrams

a) Description of main interaction with environment:

While I was designing the traffic light in smart, I focus my research on three basics scenario:

A. Normal case

B. Pedestrian crossing road case

C. Emergency car case

For the normal case as shown in Fig. 14, the car cannot cross the road cause the red light is on, so it sends a request to the traffic light to cross the road. Traffic light transfers the signal & number of vehicles in the road to the server. The server gathers all data & sends it to the traffic light algorithm who is responsible of traffic flow & handling events in the city. The algorithm is in the city's main ECU. The main ECU asks two questions sequentially. The first one is request from the server if there is an accident happened so to clear the way for emergency car. The second one is waiting time with respect to number of cars & traffic flow. The server sends the required data to the traffic light algorithm. If everything is okay, the main ECU sends approval to server which send to traffic light.

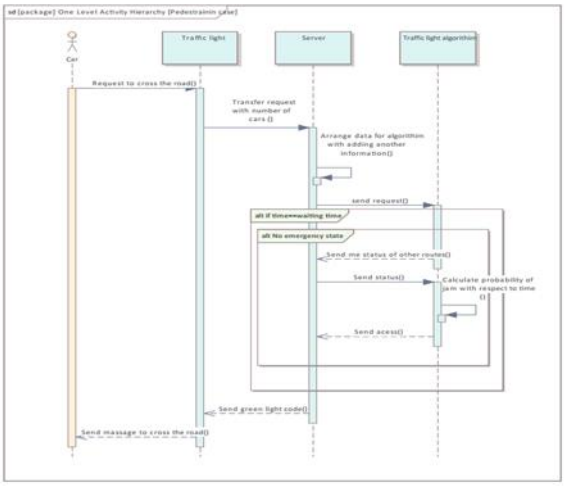


Fig. 14: shows the sequence of normal car crossing the road

The second case as shown in Fig. 15 is for pedestrians trying to cross the road. The pedestrian requests to cross the road by pushing the button near to traffic light. The traffic light turns on the alert method which are in our scenario is the buzzer & led for not crossing road. Signal transmitted to the server which is responsible of handling data & organizing it then it sends information to the traffic light algorithm. The traffic light algorithm which is located in the main ECU checks if there is an emergency case or accident that happened & emergency car needs to cross the road. If everything is clear it sends the permission to the server which transmits it to the traffic light. The traffic light receives crossing permission & gives the green light to cross the road.

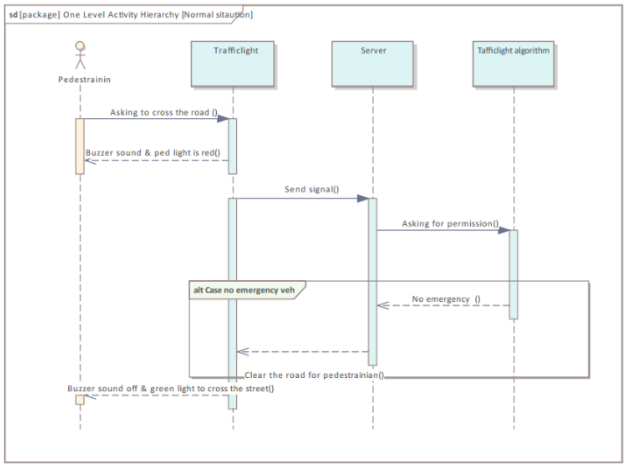


Fig. 15: shows the sequence of Pedestrian crossing the road

The third case is an emergency car as shown in Fig. 16, the emergency car has the highest priority in our design. The emergency car is driving in the emergency lane. It sends signals to the traffic light which transmits signal immediately to the server which transmits signal traffic light algorithm. The algorithm checks if he has been notified by other accidents so that the emergency car does not crash in the crossing road. Then, it asks the server where the location of accident is to calculate the fastest road to go there & make the road clear for the emergency car. It sends permission signal back to the server with route with respect to time that the car will arrive at other traffic light. The server sends a signal to the traffic light. The traffic light gives access to the emergency car for a specific amount of time with respect to the traffic light speed.

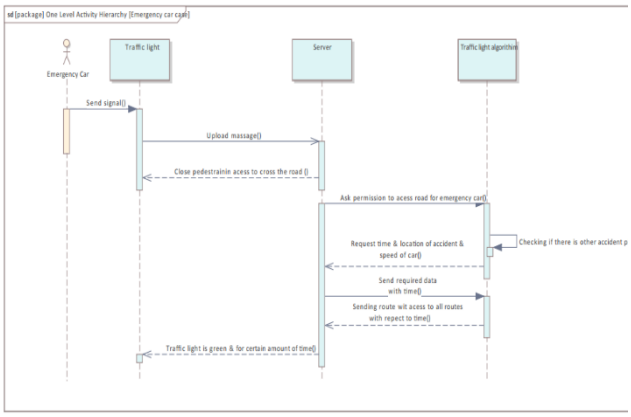


Fig. 16: shows the sequence of emergency car crossing the road

b) System constraints:

In a rapidly evolving world, the concept of a "Smart City" has gained significant attention. One crucial aspect of a Smart City is an efficient traffic control system that caters to the needs of both autonomous vehicles and pedestrians. To ensure safety and optimize traffic flow, several constraints are put in place within the system as shown in Fig. 17. The four essential constraints: Minimum Pedestrian Crossing Time, Minimum Traffic Clearance Time for Emergency Vehicles, Maximum Queue Length at Intersections, and Minimum Emergency Vehicle Response Time. These constraints contribute to creating a sustainable and intelligent urban environment, where both pedestrians and vehicles can coexist harmoniously.

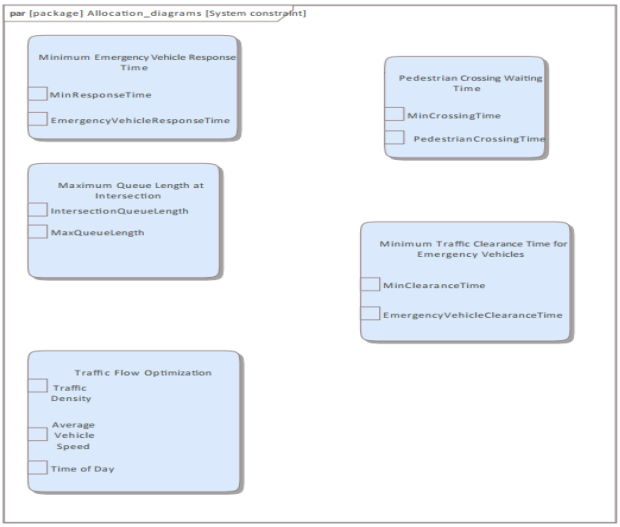


Fig. 17: shows our traffic light system constraints

Constraint 1: Minimum Pedestrian Crossing Time:
Pedestrian safety is of paramount importance in any urban environment. The "Smart City" traffic control system aims to provide sufficient time for pedestrians to safely cross the road. This constraint is defined by the parameters Minium Crossing Time and Pedestrian Crossing Time, with the

equation Pedestrian Crossing Time >= Minium Crossing Time. By allowing pedestrians adequate crossing time, the system promotes safety and reduces the risk of accidents between vehicles and pedestrians.

Constraint 2: Minimum Traffic Clearance Time for Emergency Vehicles:

Emergency vehicles require swift access through intersections to reach their destinations promptly. The system ensures that emergency vehicles are given sufficient time to clear intersections safely. The constraint is defined by the parameters Minimum Clearance Time and Emergency Vehicle Clearance Time, with the equation Emergency Vehicle Clearance Time >= Minimum Clearance Time. By allowing emergency vehicles enough clearance time, the system enables timely response to emergencies.

Constraint3: Traffic flow optimization based on traffic density, average vehicle speed, and time of day.
Constraint Equation: Optimize traffic flow by dynamically adjusting signal timings, rerouting vehicles, and providing real-time traffic information, considering factors such as traffic density, average vehicle speed, and time of day.

Constraint 4: Maximum Queue Length at Intersections:
Traffic congestion at intersections can significantly impact overall traffic flow and lead to delays. To mitigate this, the system incorporates a constraint on the maximum queue length of vehicles waiting at intersections. The constraint is defined by the parameters Maximum Queue Length and Intersection Queue Length, with the equation Intersection Queue Length <= Maximum Queue Length. By limiting the queue length, the system ensures smooth traffic flow and reduces congestion, enhancing efficiency and minimizing delays.

Constraint 5: Minimum Emergency Vehicle Response Time:
The prompt response of emergency vehicles is critical in emergency situations. The system is designed to meet a minimum response time for emergency vehicles. The constraint is defined by the parameters Minimum Response Time and Emergency Vehicle Response Time, with the equation Emergency Vehicle Response Time <= Minimum Response Time. By ensuring a quick response, the system enables emergency vehicles to reach their destinations swiftly, potentially saving lives and reducing property damage.

c) System allocation:

In our Smart City traffic control system, an essential aspect is the allocation of different subsystems and components to ensure efficient and intelligent traffic management. This allocation is represented by an allocation diagram, which visually depicts the distribution of functionalities within the system.

The allocation diagram in Fig. 18 displays how various components work together to achieve the desired outcomes. For instance, the Traffic Control Subsystem is responsible for overseeing the overall traffic flow. It is allocated to the Signal Control Algorithm, which intelligently controls the traffic signals based on real-time data.

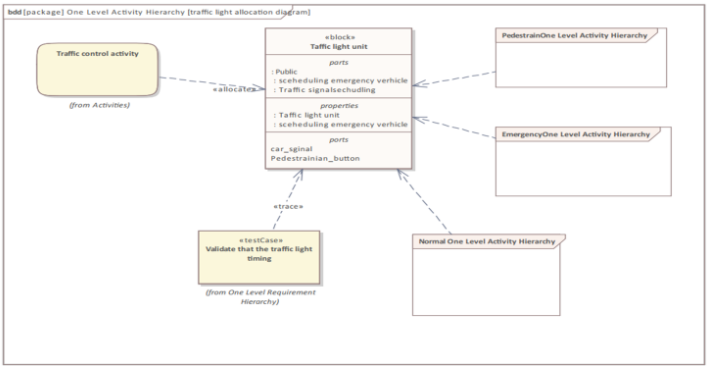


Fig. 18: shows allocation diagram integration of the system.

To gather this data, the system relies on Traffic Data Collection Sensors, which are strategically placed throughout the city. These sensors collect information on traffic density, vehicle speeds, and other relevant parameters.
Emergency Vehicle Management is another critical aspect of the system. This dedicated subsystem ensures that emergency vehicles have priority clearance at intersections when responding to urgent situations. This allocation allows emergency vehicles to navigate through traffic safely and swiftly.
Pedestrian safety is also a key consideration. The Pedestrian Crosswalk Management component is responsible for managing pedestrian crossings and ensuring their safety. It allocates resources and adjusts signal timings to provide sufficient time for pedestrians to cross the road safely.
To validate the proposed timing adjustments, the system compares the predicted traffic flow outcomes with the actual observed traffic behavior. This validation step fine-tunes the timing parameters, ensuring that the implemented changes result in improved traffic flow and reduced congestion.
With the rise of autonomous vehicles, the system allocates a dedicated Autonomous Vehicle Control component. This component oversees the control and coordination of autonomous vehicles, ensuring their seamless integration into the traffic flow while prioritizing safety.
By incorporating an allocation diagram, our Smart City traffic control system demonstrates a clear and structured distribution of responsibilities. Each component plays a specific role, working in harmony to optimize traffic flow, enhance safety, and create a more efficient transportation network.

IV. Implementation

A. *Online CAD Setup*

The system was developed utilizing TinkerCad, an online Computer-Aided Design (CAD) platform. Fig. 19 provides a snapshot of the system as visualized within the TinkerCad environment. TinkerCad offers a dedicated section known as Circuits, serving as a web-based electronic circuit simulator. This simulator specifically accommodates Arduino Uno microcontrollers, allowing users to design and test electronic circuits virtually. Notably, TinkerCad's Circuits section supports text-based code, providing users with the flexibility to program and control their simulated circuits efficiently. [3]

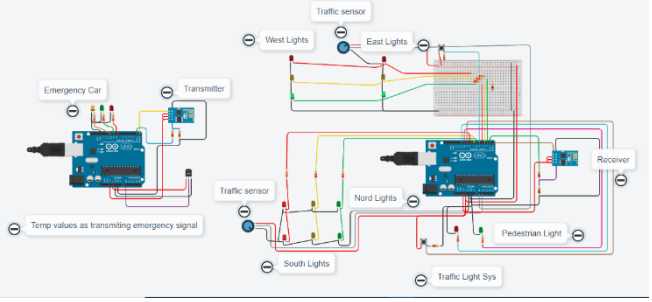


Fig. 19. Implementation of the Smart Traffic Light on TinkerCad

B. *Software Development*

The developed software was using Arduino-C on an Arduino-uno. The system developed was to simulate a smart traffic light system consisting of two intersecting roads. Each side is allowed to pass for a specified period when the light at their side is green indicating motion. The other side will intern have a red light awaiting the other vehicles to complete passing to the other side till their corresponding period for motion ends. The smart system indicated the period allowed for each side to pass depending on the number of vehicles present on each side. This can be determined in reality with the assistance of satellites detecting the number of cars present on each road with the help of GPS signals present in mobile phones and vehicles which will allow the system to calculate the congestion. This would require a real-time system with connection to the internet to know the real-time data collected from the satellites. However, the system created relies on the amount of pressure on the ground of each road to specify the congestion. This is done through the utilization of a pressure sensor called the piezo-electric transducer (Fig. 20).

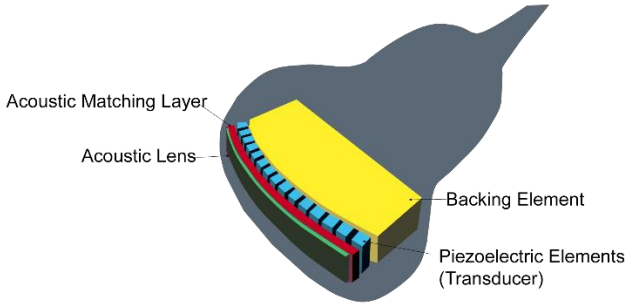


Fig. 20. pressure sensor (Piezo-electric transducer)

The mentioned sensor relies on the piezo-electric effect (Fig. 20). This is a phenomenon that occurs when some materials produce an electrical charge due to applying mechanical stress. This effect can be used in various sensors including accelerometers, force sensors and the sensor in use, pressure sensor. This allows the system to be self-sufficient in terms of data required for the optimized technique of calculating the period of operation of each road.

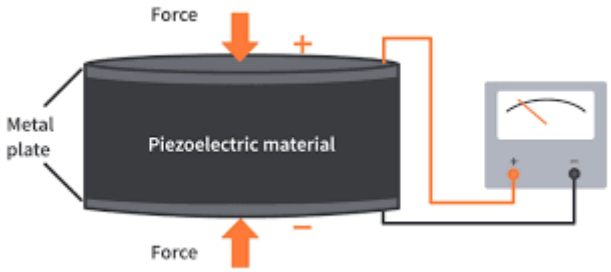


Fig. 20. Piezoelectric circuit

Another feature was the addition of an override option for emergency services like an ambulance. In this case, the priority should be to go to the lane with the ambulance. This was implemented using a push button to imitate the arrival of the ambulance requesting the passage. The system then takes the request and gives a green light to the corresponding road and a red light to the other side to allow the passage.

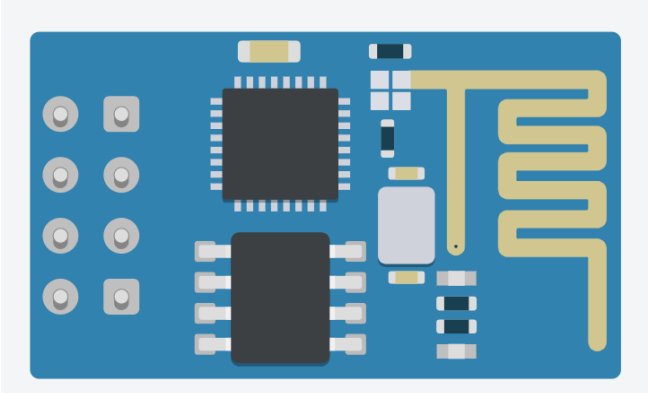


Fig. 21. ESP8266 module

To implement this feature, another Arduino-uno was used to simulate the existence of another system sending a request for priority. The connection was established between the two Arduinos using an ESP8266 wifi module (Fig. 21) on each side for connection. A push button is present at the ambulance side in order to give a signal with its arrival asking priority.

Description of Arduino Traffic Signal Simulation Code

This Arduino code as shown in Fig. 22 simulates a traffic signal system for two roads, allowing for dynamic control of signal durations through potentiometers. The setup involves LEDs representing traffic lights for each road, and the

duration of each light is determined by the analog readings from corresponding potentiometers.

The logic behind the code just covers some points (Fig. 23):

1. Compares the potentiometer readings to decide which road has higher traffic (potValueRoad1 > potValueRoad2).

2. The code then controls the traffic lights accordingly.

3. If Road 1 has higher traffic, it initiates a sequence where Road 2 is given a red light for one second, followed by a two-second yellow light, and then a green light for the duration mapped from potValueRoad1.

4. If Road 2 has higher traffic, a similar sequence is executed for Road 1.

```
void setup() {
  pinMode(redPinRoad1, OUTPUT);
  pinMode(yellowPinRoad1, OUTPUT);
  pinMode(greenPinRoad1, OUTPUT);

  pinMode(redPinRoad2, OUTPUT);
  pinMode(yellowPinRoad2, OUTPUT);
  pinMode(greenPinRoad2, OUTPUT);

  Serial.begin(9600);
}
void loop() {
  int potValueRoad1 = analogRead(potPinRoad1);
  int potValueRoad2 = analogRead(potPinRoad2);

  int durationRoad1 = map(potValueRoad1, 0, 1023, 5000, 15000);
  int durationRoad2 = map(potValueRoad2, 0, 1023, 5000, 15000);

  Serial.print("Road 1 Duration: ");
  Serial.println(durationRoad1);
  Serial.print("Road 2 Duration: ");
  Serial.println(durationRoad2);

  if (potValueRoad1 > potValueRoad2) {
    digitalWrite(redPinRoad2, HIGH);
    digitalWrite(yellowPinRoad2, LOW);
    digitalWrite(greenPinRoad2, LOW);
    delay(1000);

    digitalWrite(redPinRoad1, LOW);
    digitalWrite(yellowPinRoad1, HIGH);
    digitalWrite(greenPinRoad1, LOW);
    delay(2000);

    digitalWrite(redPinRoad1, LOW);
    digitalWrite(yellowPinRoad1, LOW);
    digitalWrite(greenPinRoad1, HIGH);
    delay(durationRoad1);
  } else {
    digitalWrite(redPinRoad1, HIGH);
    digitalWrite(yellowPinRoad1, LOW);
    digitalWrite(greenPinRoad1, LOW);
    delay(1000);

    digitalWrite(redPinRoad2, LOW);
    digitalWrite(yellowPinRoad2, HIGH);
    digitalWrite(greenPinRoad2, LOW);
    delay(2000);
```

Fig. 22. Arduino code of the implemented circuit

Also Here's a state machine diagram to illustrate the logic:

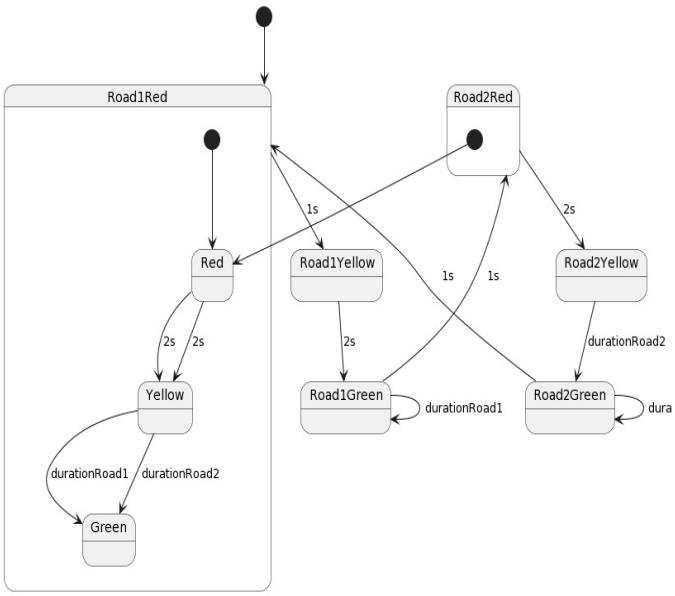


Fig. 23. State Machine illustrates the logic behind the smart traffic

C. *Deployment*

This section focuses on the rigorous validation and verification of a simulation-based system by converting its code to pure C++ for testing purposes. Ensuring the accuracy and dependability of the deployed system is the main goal. Several test cases are run on the system using Google Test, a popular C++ testing framework.

After the system is fully implemented, testing is done. This emphasizes how important it is to validate and verify the code that has been produced. To enable comprehensive testing with the Google Test framework, the entire codebase is changed to pure C++, as illustrated in Fig. 24

```
}

void TrafficController::setup() {
    std::cout << "Simulating pinMode" << std::endl;
}

void TrafficController::loop(int potValueRoad1, int potValueRoad2) {
    int durationRoad1 = map(potValueRoad1, 0, 1023, 5000, 15000);
    int durationRoad2 = map(potValueRoad2, 0, 1023, 5000, 15000);

    std::cout << "Road 1 Duration: " << durationRoad1 << std::endl;
    std::cout << "Road 2 Duration: " << durationRoad2 << std::endl;

    if (potValueRoad1 > potValueRoad2) {
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));

        std::this_thread::sleep_for(std::chrono::milliseconds(2000));

        std::this_thread::sleep_for(std::chrono::milliseconds(durationRoad1));
    } else {
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));

        std::this_thread::sleep_for(std::chrono::milliseconds(2000));

        std::this_thread::sleep_for(std::chrono::milliseconds(durationRoad2));
    }

    std::cout << "Simulating digitalWrite for turning off all LEDs" << std::endl;
}
```

Fig. 24. The converted CPP code of the implementation

Seven comprehensive test cases are created and run on the C++ code, as shown in Fig. 25. Also as can be seen in Figure 3.3.1, the result shows that every test case passes, confirming the system's initial robustness.

```
TEST_F(TrafficControllerTest, SetupFunction) {
    controller.setup();
}

TEST_F(TrafficControllerTest, MapWrapperFunction) {
    int result = controller.mapWrapper(512, 0, 1023, 5000, 15000);
    EXPECT_EQ(result, 10004);
}

TEST_F(TrafficControllerTest, LoopFunctionWithZeroDuration) {
    int potValueRoad1 = 0;
    int potValueRoad2 = 0;
    controller.loop(potValueRoad1, potValueRoad2);
}

TEST_F(TrafficControllerTest, LoopFunctionWithEqualPotValues) {
    int potValueRoad1 = 512;
    int potValueRoad2 = 512;
    controller.loop(potValueRoad1, potValueRoad2);
}

TEST_F(TrafficControllerTest, LoopFunctionWithPot1GreaterThanPot2) {
    int potValueRoad1 = 800;
    int potValueRoad2 = 400;
    controller.loop(potValueRoad1, potValueRoad2);
}
```

Fig. 25. sample of the seven test cases

Here's the Seven test cases performed on the implementation:

MapFunction:
Purpose: Verify the correct functionality of the map function.
Testing Approach: Assess the mapping of different input values and ensure the output aligns with the expected results.

LoopFunction:
Purpose: Test the overall behavior of the loop function.
Testing Approach: Evaluate diverse scenarios, such as varying pot values for two roads, to validate the function's output, including considerations for delays and LED control.

SetupFunction:
Purpose: Focus on the setup function and simulate pinMode.
Testing Approach: Ensure the setup function executes without errors and adheres to the expected behavior.

MapWrapperFunction:
Purpose: Verify the correctness of the mapWrapper function.
Testing Approach: Utilize this utility function for testing, checking if mapping is performed accurately based on specified parameters.

LoopFunctionWithZeroDuration:
Purpose: Test the loop function with both road durations set to zero.
Testing Approach: Ensure the function handles zero duration scenarios appropriately.

LoopFunctionWithEqualPotValues:
Purpose: Check the loop function when both pot values are equal.
Testing Approach: Confirm the system behaves correctly under conditions where both roads experience the same traffic conditions.

LoopFunctionWithPot1GreaterThanPot2:
Purpose: Focus on the loop function when the pot value of road 1 is greater than road 2.
Testing Approach: Evaluate how the system manages uneven traffic conditions.

Five more defect test cases, as depicted in Fig. 26, are added to the system to further confirm its robustness. As demonstrated in Figure 3.3.1, three of the five defect test cases fail, exposing possible weaknesses and flaws in the system.

```
59
60  TEST_F(TrafficControllerTest, MapWrapperNegativeInput) {
61      int result = controller.mapWrapper(-10, 0, 100, 0, 200);
62      EXPECT_LT(result, 0);
63  }
64  TEST_F(TrafficControllerTest, IncorrectLoopLEDOff) {
65      int potValueRoad1 = 700;
66      int potValueRoad2 = 300;
67      EXPECT_NE(controller.mapWrapper(potValueRoad1, 0, 1023, 5000, 15000),
68                controller.mapWrapper(potValueRoad2, 0, 1023, 5000, 15000));
69  }
70  TEST_F(TrafficControllerTest, IncorrectMapWrapperBehavior) {
71      int potValue = 750;
72      int in_min = 500;
73      int in_max = 1000;
74      int out_min = 2000;
75      int out_max = 5000;
76
77      int incorrectResult = controller.mapWrapper(potValue, in_min, in_max, out_max, out_min);
78
79      EXPECT_NE(incorrectResult, controller.mapWrapper(potValue, in_min, in_max, out_min, out_max));
80  }
81  TEST_F(TrafficControllerTest, IncorrectMapWrapperOutOfBounds) {
82      int potValue = 1100;
83      int in_min = 0;
84      int in_max = 1000;
85      int out_min = 2000;
86      int out_max = 5000;
87
88      EXPECT_LT(controller.mapWrapper(potValue, in_min, in_max, out_min, out_max), out_min);
89  }
90
91
```

Fig. 26. sample of the 5 defect cases

Here is the description of the defect test cases done on the code itself:

IncorrectLoopEqualPotValues:
Purpose: Introduce an intentional failure by checking if the mapWrapper results for equal pot values are not equal.
Testing Approach: Identify potential defects in how the system handles equal traffic conditions.

MapWrapperNegativeInput:
Purpose: Test the mapWrapper function with a negative input.
Testing Approach: Ensure the function correctly handles negative values, with the expectation set for a negative result.

IncorrectLoopLEDOff:
Purpose: Intentionally introduce a failure by checking if the LED turn-off behavior in the loop function is incorrect for specific pot values.
Testing Approach: Identify defects in the LED control mechanism.

IncorrectMapWrapperBehavior:
Purpose: Catch defects in the mapWrapper function by intentionally providing incorrect mapping parameters.
Testing Approach: Confirm that the result is not equal to the correct mapping, exposing potential flaws.

IncorrectMapWrapperOutOfBounds:
Purpose: Test the mapWrapper function with a pot value outside the specified range.
Testing Approach: Verify that the function handles out-of-bounds input correctly, addressing potential issues with input validation.

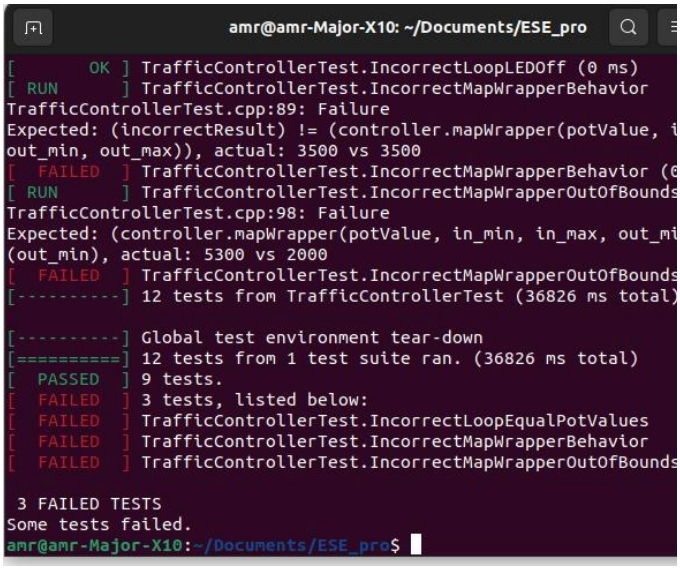


Fig. 27. The results of the test cases.

Subsequently, the code is revisited and refined to address the identified defects. The corrective development process ensures that the system is fortified against potential issues, thereby enhancing its overall reliability.

D. Challenges and Solutions

The first challenge that occurred was discovering the limitations of the used simulation environment ThinkerCad where not all the required components were found on its environment. This led to the improvised version of the project that used a substitution firstly for the piezoelectric transducer. The replaced component was an essential part of the project. This led to choosing an alternative that would imitate the effect. However, without the use of the sensor being placed on the simulated environment. The component used was a variable resistor where two variable resistors (Fig. 28) were placed to imitate the congestion on each road. The higher the resistance of the variable resistor, the greater the congestion at the corresponding road. The output of both variable resistors is then compared. The difference between the resistances is converted into differences between the traffic congestions between the two roads which intern changes the ratio between the allowance of each road for passage giving the more crowded road an increased amount of time for passage.

Fig. 28. Variable resistor

This lead another improvised solution in this project which is the utilization of UART communication protocol in Arduinos. This protocol uses the build in RX and TX pins in the Arduino to perform communication between them.

Since some years, the ESP8266 Wi-Fi is not supported anymore on TinkerCad. So, to go around this limitation, we took an old project where two Wi-Fi modules were present. We only kept these components, built our circuits and easily inserted them.
We used two ESP8266 Wi-Fi modules. One, the transmitter, for the circuit simulating the ambulance and the other, the receiver, on the traffic light circuit.
Another challenge was to find a suitable signal to send from the emergency vehicle to the traffic and how. The cast fell on temperature. The emergency vehicle will send two types of temperature numbers. Those below 50 degrees Celsius would be ignored. However, those egal or greater than 50 degrees Celsius will trigger a change of state of the traffic light. In our implementation, we only focused on the North-South traffic lights. So, this traffic light will turn green for a longer time. And as long as the triggering signal will be present, then the traffic will also be green longer.
The last challenge was to find a suitable medium to exchange data between the two ESP8266 wifi modules. We tried a direct connection, but we failed. Finally, we used a pretty good alternative: Thingspeak, unfortunately, we also added delays in the transmitted signals. But the good news is that everything worked flowless.

V. Evaluation and Discussion

A. Scheduling

In the implementation, the Arduino-Uno that was used provides data which is not accurate for simulating a real-time system. This led to the use of another code for implementing the scheduling on the smart traffic light system.

The scheduling applied for the system was implemented using both EDF and RMS. It's done with having three tasks; the first road, the second road, and the ambulance treated as a third task for simplicity.

```python
import time
import random
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import pandas as pd

class Task:
    def __init__(self, name, period, deadline, execution_time):
        self.name = name
        self.period = period
        self.deadline = deadline
        self.next_release_time = time.time() + period
        self.execution_time = execution_time
        self.execution_times = []

    def execute(self, current_time):
        self.execution_times.append(current_time)
        current_time_seconds = int(current_time)  # Extracting the integer part (seconds)
        last_three_digits = current_time_seconds % 1000  # Extracting the last three digits
        print(f"{self.name} is executing at time {last_three_digits}")

    def update_next_release_time(self):
        self.next_release_time += self.period
```

Fig. 29. Showing the Class Task with Attributes

In this code, the scheduling is performed as follows. the periodic task in real-time system is presented as a class named task. The class task has the following attributes.
name, period, deadline, excution_time, next_release_time which is the time where the task will be available next and finaly execution_times which is a list containing all the times where tasks are executed during the program's simulation.

The methods here are two methods which are firstly execute(current_time) printing out a notification that says that the task is being excuted at the corresponding time period. The second method called update_next_release_time() its job is to update the new arrival time for the task as from which it will start again in the following period.

```python
def visualize_execution(tasks, num_periods):
    replicated_tasks = []
    for i in range(1, num_periods):
        replicated_tasks.extend(
            {
                "Task": task.name,
                "Start": task.execution_times[-1],
                "Finish": task.execution_times[-1] + task.period,
            }
            for task in tasks
        )
    df = pd.DataFrame(replicated_tasks)
    bar_height = 0.1

    fig, ax = plt.subplots(figsize=(10, bar_height * len(df["Task"].unique())))

    ax.set_yticks([i * bar_height + bar_height / 2 for i in range(len(df["Task"].unique()))])
    ax.set_yticklabels(df["Task"].unique())

    for i, task in enumerate(df["Task"].unique()):
        task_df = df[df["Task"] == task]
        ax.broken_barh(
            [(start, finish - start) for start, finish in zip(task_df["Start"], task_df["Finish"])],
            (i * bar_height, bar_height),
            facecolors=["blue", "orange", "green"],
        )

    ax.set_xlabel("Time")
    ax.set_title(f"Scheduling Visualization ({num_periods} Periods)")

    plt.show()
```

Fig. 30. Showing the Visualization code

Visualization was implemented using the function visualize_execution. In this function, two libraries are utilized which are Matplotlib and pandas data frame that was used to organize the data needed. It takes as a input the list of tasks and the required period to be observed.

```python
def edf_scheduler(tasks, simulation_time):
    start_time = time.time()

    while time.time() - start_time < simulation_time:
        current_time = time.time()
        ready_tasks = [task for task in tasks if current_time >= task.next_release_time]

        if ready_tasks:
            earliest_deadline_task = min(ready_tasks, key=lambda task: task.deadline)
            earliest_deadline_task.execute(current_time)
            earliest_deadline_task.update_next_release_time()

        for task in tasks:
            if task.next_release_time <= current_time:
                task.execute(current_time)
                task.update_next_release_time()

        time.sleep(1)

    visualize_execution(tasks, int(simulation_time / min(task.period for task in tasks)))
```

Fig. 31. Showing the EDF Scheduler code

Earliest Deadline First scheduling algorithm is implemented using the edf_scheduler fumction. It checks on the tasks to find the ones which their release time has come in order to add them to the scheduler and the task with the earliest deadlibne is executed first from the queue of tasks and updates the arrival of the task in the following period then sleeps for 1 second to simulate time passing in real life.

```python
def rms_scheduler(tasks, simulation_time):
    start_time = time.time()

    while time.time() - start_time < simulation_time:
        current_time = time.time()
        ready_tasks = [task for task in tasks if current_time >= task.next_release_time]

        if ready_tasks:
            highest_priority_task = min(ready_tasks, key=lambda task: task.period)
            highest_priority_task.execute(current_time)
            highest_priority_task.update_next_release_time()

        for task in tasks:
            if task.next_release_time <= current_time:
                task.execute(current_time)
                task.update_next_release_time()

        time.sleep(1)

    visualize_execution(tasks, int(simulation_time / min(task.period for task in tasks)))
```

Fig. 32. Showing the RMS scheduler implementation

Rate Monotonic Scheduling algorithm is implemented using the function rms_scheduler and its implemented similarly to the EDF but with the higher priority going to the task with shortest period. Followed by that are two main functions to simulate each scheduler.

```python
def main_edf():
    road1_light = Task("Traffic Light Road 1", 20, 20, 7)
    road2_light = Task("Traffic Light Road 2", 20, 20, 7)
    ambulance = Task("Ambulance", 80, 10, 5)

    tasks = [road1_light, road2_light, ambulance]

    edf_scheduler(tasks, simulation_time=300)

def main_rms():
    road1_light = Task("Traffic Light Road 1", 20, 20, 7)
    road2_light = Task("Traffic Light Road 2", 20, 20, 7)
    ambulance = Task("Ambulance", 80, 10, 5)

    tasks = [road1_light, road2_light, ambulance]

    rms_scheduler(tasks, simulation_time=300)

if __name__ == "__main__":
    main_edf()
    main_rms()
```

Fig. 33. Showing the Main methods

- *Analysis*

```python
class Task:
    def __init__(self, name, wcet, period, deadline=float('inf')):
        self.name = name
        self.wcet = wcet
        self.period = period
        self.deadline = deadline

def calculate_analysis(tasks):
    for task in tasks:
        wcrt = task.wcet
        bwcrt = task.wcet
        q_wcet = task.wcet

        for other_task in tasks:
            if other_task != task:
                q_wcet += (1 / other_task.period) * other_task.wcet
                bwcrt += (q_wcet - task.wcet) * other_task.wcet

        print(f"Analysis of Result:")
        print(f"• {task.name} in priority {task.period}: wcrt={wcrt}")
        print(f"bwcrt=q*WCET:{task.period}*{bwcrt}={bwcrt}")
        print(f"q*WCET:{task.period}*{q_wcet}={q_wcet}\n")

# Define the tasks
road1_light = Task("Traffic Light Road 1", 7, 20, 20)
road2_light = Task("Traffic Light Road 2", 7, 20, 20)
ambulance = Task("Ambulance", 5, 15, 80)  # Assuming a WCET of 2 for the ambulance

# Perform the analysis
calculate_analysis([road1_light, road2_light, ambulance])
```

Fig. 34. Showing the Analysis code

The analysis was done through a code imitating the principles of the PYCPA library to calculate the worst-case execution time as well as the bwcrt which is the blocking time. The analysis obtained valid results on the output of the tasks scheduled

```
• Traffic Light Road 1 in priority 20: wcrt=7
bwcrt=q*WCET:20*12.86666666666666=12.86666666666666
q*WCET:20*7.683333333333333=7.683333333333333

Analysis of Result:
• Traffic Light Road 2 in priority 20: wcrt=7
bwcrt=q*WCET:20*12.86666666666666=12.86666666666666
q*WCET:20*7.683333333333333=7.683333333333333

Analysis of Result:
• Ambulance in priority 15: wcrt=5
bwcrt=q*WCET:15*12.349999999999993=12.349999999999993
q*WCET:15*5.699999999999999=5.699999999999999

PS C:\Users\Work pc\Desktop\SysML>
```

Fig. 35. Showing the results of the analysis

B. Inspection

1) Inspection report:

Overview of Inspected Document:

The inspected document, titled "Traffic Light Control System Design," outlines the specifications and design considerations for the traffic light control system within the Smart City infrastructure. The document aims to provide insights into the functionality, architecture, and constraints of the traffic light control system.

Positive aspects:

- **clear functional requirement:**

The documents effectively outline the functional requirements of the traffic light control system, including traffic flow optimization, emergency vehicle prioritization, and pedestrian safety.

- **consideration of emergency scenarios:**

Emergency vehicle prioritization is appropriately addressed, demonstrating an understanding of the importance of ensuring the timely passage of emergency vehicles.

- **Integration with smart city infrastructure:**

The documents highlight the integration of the traffic light control system with other Smart City components, fostering an integrated approach to urban mobility.

- **Adaptability to varying traffic conditions**:

The documents demonstrate an understanding of the dynamic nature of traffic conditions like environmental change or weather change. The inclusion of adaptive strategies that allow the system to respond to changing traffic patterns and unexpected events positively contributes to system resilience.

- **Optimized traffic flow algorithms:**

The documents highlight well-thought-out algorithms for optimizing traffic flow at intersections. The inclusion of intelligent traffic signal control mechanisms is a strong positive aspect, ensuring efficient and dynamic traffic management.

- **Failure Handling and Redundancy condition**

Taking into consideration maximum allowable downtime in the event of a failure to restore system back. Taking into consideration Hazards may face the system like power supply & backup controllers.

Negative aspects:

- **Dependent need on using sensors**.

The document shows that we are using just sensors to detect pedestrians but in case of component failure in the system would case a hug trouble.

- **Lack of Redundancy Strategy:**

In making the system all possible maintenance in case of component failures should be documented & while launching new version of the system there should be communication protocol for updating the system

- **Lack for VMS variable in the requirements of system**

There should be VMS display that notifies driver of the upcoming accident that might happen or traffic congestion.

Influence on an Updated Version:

- Inclusion of redundancy strategy

An updated version of the document should include a section detailing the redundancy strategy for critical components, ensuring the reliability and availability of the traffic light control system.

- **Replace sensor strategy.**

It is better to replace the ultrasonic sensor with motion sensors & 360-degree camera. It will enhance the reliability of the system. Even when one of the sensors breaks down, the camera can support it.

- **Adding protocol for updating the system wirelessly**

Applying Flashing over the air concepts (FOTA), to update the system even in long distances.

C. Fault tree Analysis

Urban traffic management heavily depends on traffic light systems to ensure the safe and smooth movement of vehicles and pedestrians at intersections. While these systems are crucial, they are not perfect and can face issues. This document takes a close look at the problems traffic lights might encounter, requiring a careful examination of their weaknesses.

In simpler terms, we'll be exploring the potential problems traffic lights could have and figuring out what might cause them to not work as expected. This understanding is important for keeping our roads safe and traffic flowing well.

Looking into potential problems with traffic lights, we'll pinpoint key concerns. We'll explore issues like power outages, sensor problems, software glitches, communication breakdowns, and maintenance lapses. By understanding these problems, our goal is to help decision-makers take smart actions to prevent issues and set up strong maintenance plans.

In the upcoming sections, we'll break down each problem. We'll create a detailed fault tree (Fig. 36) to show how different events can lead to traffic light failures. Adding in probabilities will give a clear idea of how likely different issues are, helping us focus on the most important ones when planning ways to reduce risks.

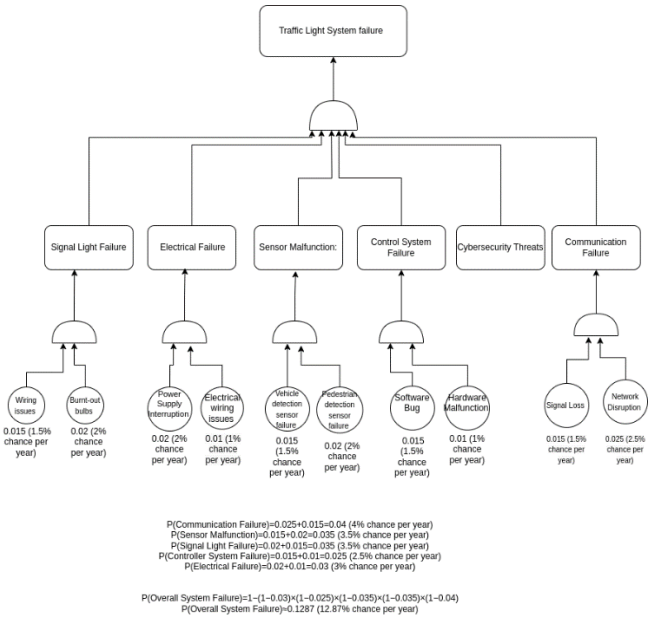


Fig. 36. Fault Tree

- **Signal Light Failure:**

The hazard of signal light failure encompasses situations where the visual indicators of the traffic lights malfunction, leading to potential confusion among road users. This hazard could result from various factors, such as bulb burnouts, wiring issues, or other mechanical failures within the signal light system.

- **Electrical Failure:**

Electrical failure represents a hazard associated with disruptions in the power supply or related electrical components supporting the traffic light infrastructure. This could result from power grid failures, electrical component malfunctions, or other issues impacting the electrical integrity of the system.

- **Sensor Malfunction:**

Sensor malfunction introduces the risk of inaccuracies in detecting the presence of vehicles or pedestrians at intersections. Environmental interference, sensor calibration errors, or other issues may contribute to this hazard.

- **Control System Failure:**

The hazard of control system failure involves the breakdown of the central system responsible for coordinating and managing the entire traffic light network. This could result from coding errors, software glitches, or failures during system updates.

- **Cyber Security Threats:**

Cybersecurity threats present a modern hazard where the traffic light system becomes susceptible to malicious attacks or unauthorized access. These threats could compromise the integrity of the control system, leading to unauthorized manipulation of traffic signals.

- **Communication Failure:**

Communication failure involves disruptions in the data transmission between different components of the traffic light system. This hazard could result from network outages, data transmission errors, or other issues impeding the seamless communication required for synchronized traffic control. Assigning probabilities to identified hazards in traffic light systems is a critical aspect of risk assessment. Probabilities provide a quantitative measure of the likelihood of each hazard occurring, aiding decision-makers in prioritizing mitigation efforts.

D.2 Handling Hazards in Traffic Light Systems:

Traffic light systems are pivotal for urban traffic management, yet vulnerabilities exist that may

compromise their functionality and safety. We aim to offer a comprehensive framework to handle identified hazards, focusing on both the design and implementation phases of traffic light systems.

D.2.1 Signal Light Failure:

- *Design Level:*
  - Incorporate redundant signal lights to ensure continuous functionality.
  - Implement automated monitoring systems to detect failures promptly.
- *Implementation Level:*
  - Regularly inspect and maintain signal lights, replacing bulbs and addressing wiring issues promptly.
  - Establish a user feedback mechanism to report malfunctioning signal lights.

D.2.2 Electrical Failure:

- *Design Level:*
  - Integrate surge protection and backup power systems to withstand power fluctuations.
  - Design electrical components with durability and reliability in mind.
- *Implementation Level:*
  - Regularly inspect and maintain electrical components, addressing signs of wear or degradation proactively.
  - Implement a proactive power outage detection and response system.

D.2.3 Sensor Malfunction:

- *Design Level:*
  - Employ multiple sensor technologies for enhanced reliability.
  - Implement self-diagnostic features in sensors to detect and report malfunctions.
- *Implementation Level:*
  - Regularly calibrate sensors and replace faulty units during routine maintenance.
  - Implement real-time monitoring with alert systems to address potential sensor malfunctions promptly.

D.2.4 Control System Failure:

- *Design Level:*
  - Utilize fault-tolerant design principles in the control system architecture.
  - Implement comprehensive testing procedures for software updates.
- *Implementation Level:*
  - Regularly update and patch software, ensuring compatibility and stability.
  - Develop and regularly test backup and recovery plans for the control system.

D.2.5 Cyber Security Threats:

- *Design Level:*
  - Implement encryption and secure communication protocols to safeguard against cyber threats.
  - Incorporate access controls and robust authentication mechanisms.
- *Implementation Level:*
  - Regularly update and patch software to address vulnerabilities.
  - Conduct regular cybersecurity audits and penetration testing to identify and rectify potential threats.

D.2.6 Communication Failure:

- *Design Level:*
  - Utilize redundant communication channels to minimize the impact of communication failures.
  - Implement error-checking mechanisms in data transmission.
- *Implementation Level:*
  - Regularly test and monitor communication channels for reliability.
  - Establish contingency plans for alternative communication routes in case of failures.

VI. Conclusion

In this paper we have delineated the process of developing a prototype for our system, which involved refining the system requirements and designing a traffic light system for a smart city. This process also included simulating the system's behavior for partial verification and validation. We have considered potential hazards to enhance the system's safety. Our design specifically focused on three main scenarios: A normal car requesting to cross the road, a pedestrian requesting to cross the road, an emergency vehicle, which has the highest priority, requesting to cross the road. Despite our efforts, we acknowledge that our design has limitations due to the vast number of domains that need exploration, which is beyond our current capacity. For instance, our system does not adequately address issues such as communication failure and cybersecurity to safeguard against malware attacks. However, our primary focus remains on our approach to system construction. Looking ahead, we anticipate the involvement of numerous stakeholders in this project. Their contributions will be invaluable in refining the system requirements in line with our system constraints, thereby enhancing the design of the system and making it more reliable & dependable.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Prof Dr. Prof. Stefen Henkler, from FH Dortmund, for his exceptional teaching, continuous guidance, insightful suggestions, comments and support throughout the semester

REFERENCES

[1] Delligatti, L., 2013. SysML distilled: A brief guide to the systems modeling language. Addison-Wesley.

[2] J. Muvuna, T. Boutaleb, K. J. Baker, and S. B. Mickovski, "A Methodology to Model Integrated Smart City System from the Information Perspective," Smart Cities, vol. 2, no. 4, pp. 496–511, Nov. 2019, doi: 10.3390/smartcities2040030.

[3] Wikipedia, website accessed 30 January 2023 < https://en.wikipedia.org/wiki/Tinkercad >

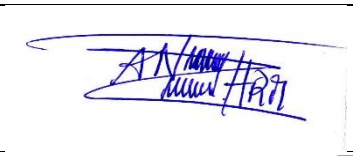
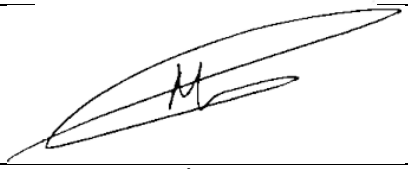
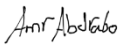
[4] Tinkercad, website accessed 30 January 2023, <https://www.tinkercad.com/things/ecgvRQPwoBC-smart-traffic-lights/editel?returnTo=%2Fdashboard>

AFFIDAVIT

We (Anguiga Hermann, Ismail Elsayed, Hadis Mohammadi, Raed Kayali, Amr Abdelaziz) Herewith declare that:

We have composed the present paper and work ourselves without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted are marked as such other references about the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in art used in another examination or as a course performance.

Date: 30.01.2024

Name	Signature
Anguiga Hermann	
Ismail Elsayed	
Hadis Mohammadi	
Raed Kayali	
Amr Abdelaziz	

ANNEX

CONTRIBUTION

Member Full Name	**Contribution in Percentages**	**Work Estimation in Hours**	**Parts written on this paper**
Anguiga Hermann	20%	40H	**Introduction**, **Design Approach** (SysML Overview, SysML in Smart city design, Requirements Analysis, System Architecture Design, use case diagram, context diagram), **Design of Smart Traffic Light System** (System Architecture, Key Components), **Implementation** (, Challenges, and solutions).
Ismail Elsayed	30%	50H	Drawing diagrams, Activity diagrams, Sequence diagram, Constraints diagram, allocation diagram, Inspection report, Conclusion
Hadis Mohammadi	10%	25H	Fault tree Analysis, fault tree diagram
Raed Kayali	20%	40 H	Scheduling and Implementation
Amr Abdelaziz	20%	40 H	Implementation, testing, state machine diagram, Abstract

Complementary Information

Project design material

- https://github.com/IsmailosamaFathy/ESE_project

Implementation and simulation

- https://www.tinkercad.com/things/ecgvRQPwoBC-smart-traffic-lights/editel?returnTo=%2Fdashboard